En primer lugar modificamos nuestro POJO incluyendo en él una variable boolean para saber si dicho elemento ha sido seleccionado o no

```
public class Usuario {
     String nombre;
     String apellido;
     String correo;
     Boolean seleccionado;
```

La implementación del modo de selección múltiple implica que este se activará ante la pulsación larga sobre un elemento del recycler. Una vez activado, el click sobre cualquier elemento del recycler permitirá la selección o no del elemento pulsado. Este código va en la activity donde se encuentra el recycler.

Veamos en primer lugar la gestión del click sobre un elemento del recycler:

```
adaptador.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int posicion=recyclerView.getChildAdapterPosition(v);
        if(actionMode!=null) {
            if (intercambiarSeleccion(posicion)) datos.get(posicion).seleccionado = true;
            else datos.get(posicion).seleccionado = false;
        }
        else Toast.makeText( context: MainActivity.this,  text: "Has pulsado" + recyclerView.getChildAdapterPosition(v), Toast.LENGTH_SHORT).show();
    } });
```

Si el actionMode no está activado es una pulsación normal y en este caso mostramos un toast, pero podría ser cualquier otra acción específica de nuestra aplicación.

Si está activado entonces hay que seleccionar o deseleccionar el ítem pulsado según sea su estado. Esto lo hacemos invocando al método intercambiarSeleccion() y actualizando el valor seleccionado del objeto que hemos pulsado.

Y ahora veamos la gestión del click largo sobre un elemento del recycler:

```
adaptador.setOnLongClickListener(new View.OnLongClickListener() {
    @Override
    public boolean onLongClick(View v) {
        int posicion=recyclerView.getChildAdapterPosition(v);
        if(actionMode==null)
            actionMode=startSupportActionMode(actionModeCallback);
        if(intercambiarSeleccion(posicion)) datos.get(posicion).seleccionado=true;
        else  datos.get(posicion).seleccionado=false;
        return false;
    }
});
```

Se supone que esta acción solo desencadena la activación de nuestro selector múltiple de elementos, es decir si está desactivado invocamos a nuestro actionModeCallback (mostrar el menú CAB) y posteriormente gestionamos la selección o no del elemento pulsado.

Veamos el método intercambiarSeleccion()

```
private boolean intercambiarSeleccion(int posicion) {
    boolean seleccionado=adaptador.toggleSelection(posicion);
    int count=adaptador.getSelectedItemCount();
    if(count==0) actionMode.finish();
    else{
        actionMode.setTitle(String.valueOf(count));
        actionMode.invalidate();
    }
    return  seleccionado;
}
```

actionMode.invalidate actualiza el contenido del menú.

Tenemos que modificar el holder de nuestro adaptador para aplicar el efecto deseado cuando se realiza la selección

```
@RequiresApi(api = Build.VERSION_CODES.M)
public void bind(Usuario usuario, int pos){
    if(usuario.seleccionado)v.setBackgroundColor( context.getResources().getColor(R.color.oscuro,  theme: null));
    else v.setBackgroundColor(ContextCompat.getColor(context,R.color.claro ));
    txtNombre.setText(usuario.getNombre());
    txtApellido.setText(usuario.getApellido());
    this.usuario=usuario;
}
```

Lo único que hacemos es modificar el color de fondo entre dos tonos distintos definidos en el archivo colors.

A través de ActionMode.Callback vamos a implementar la aparición del menú de acciones CAB, así como las acciones específicas que aparecen en el mismo.

```
class ActionModeCallback implements ActionMode.Callback {
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        mode.getMenuInflater().inflate(R.menu.menu_layout,menu);
        return true;
    }

    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false;
    }

    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
            case R.id.eliminar:
                adaptador.eliminarItemsSeleccionados(adaptador.getSelectedItems());
                mode.finish();
                return  true;
        }
        return false;
    }

    @Override
    public void onDestroyActionMode(ActionMode mode) {
        adaptador.clearSelection();
        adaptador.desactivarSeleccion();
        actionMode=null;
    }
}
```

Es muy sencillo y como vemos en él invocamos acciones implementadas en nuestro adaptador para actuar sobre el mismo: eliminarItemsSeleccioados(), clearSeleccion() y desactivarSeleccion().

Veamos el adaptador, cuya definición es muy parecida a la vista hasta ahora, la particularidad es que extiende de la clase SelecionableAdapter, que ahora después comentaremos, pero que a su vez hereda de RecyclerView.Adapter y donde desarrollamos o gestionamos el SpandableBooleanArray .

```
public class Adaptador extends SeleccionableAdapter implements View.OnClickListener, View.OnLongClickListener{
    private ArrayList<Usuario> datos;
    private View.OnClickListener listener;
    private View.OnLongClickListener listenerLong;
    private Context context;

    Adaptador(ArrayList<Usuario> datos, Context context) {
        this.datos=datos;
        this.context=context;
    }

    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        View itemView = LayoutInflater.from(viewGroup.getContext())
                .inflate(R.layout.card_view, viewGroup,  attachToRoot: false);
        itemView.setOnClickListener(this);
        itemView.setOnLongClickListener(this);
        return new Holder(itemView, context);
    }

    void eliminarItemsSeleccionados(List<Integer> items) {
        Collections.sort(items);
        Collections.reverse(items);
        ArrayList<Usuario> aux=new ArrayList<Usuario>(datos);
        for(int i:items) aux.remove(i);
        datos= new ArrayList<Usuario>(aux);
        for(int i: items) notifyItemRemoved(i);
    }

    void desactivarSeleccion() {
        for(Usuario i: datos) i.seleccionado=false;
    }
```

Además codificamos el onClick y onLongClick como siempre.

Veamos nuestro adaptador:

```
abstract class SeleccionableAdapter extends RecyclerView.Adapter{
    private SparseBooleanArray selectedItems;

    public SeleccionableAdapter() { selectedItems = new SparseBooleanArray(); }
      boolean isSelected(int position) { return getSelectedItems().contains(position); }

    boolean toggleSelection(int position) {
        boolean seleccionado;
        if (selectedItems.get(position,  valueIfKeyNotFound: false)) {//devuelve el valor boolean de si está true en
                                          // la position indicada y si no lo encuentra devuelve false
            selectedItems.delete(position);
            seleccionado=false;
        } else {
            selectedItems.put(position, true);
            seleccionado=true;
        }
        notifyItemChanged(position);
        return  seleccionado;
    }
    void clearSelection() {
        List<Integer> selection = getSelectedItems();
        selectedItems.clear();
        for (Integer i : selection) {
            notifyItemChanged(i);
        }
    }

    int getSelectedItemCount() { return selectedItems.size(); }
    List<Integer> getSelectedItems() {
        List<Integer> items = new ArrayList<>(selectedItems.size());
        for (int i = 0; i < selectedItems.size(); ++i) {
            items.add(selectedItems.keyAt(i));
        }
        return items;
    }
}
```